Wireless Networks 5 (1999) 411–427

AJ

Composable ad hoc location-based services for heterogeneous mobile clients

Todd D. Hodes and Randy H. Katz

Computer Science Division, University of California, Berkeley, CA, 94720, USA

This paper introduces a comprehensive architecture that supports adapting a client device's functionality to new services it discovers as it moves into a new environment. Users wish to invoke services – such as controlling the lights, printing locally, gaining access to application-specific proxies, or reconfiguring the location of DNS servers – from their mobile devices. But *a priori* standardization of interfaces and methods for service invocation is infeasible. Thus, the challenge is to develop a new service architecture that supports heterogeneity in client devices and controlled objects while making minimal assumptions about standard interfaces and control protocols. Four capabilities are needed for a comprehensive solution to this problem: (1) allowing device mobility, (2) augmenting controllable objects to make them network-accessible, (3) building an underlying discovery architecture, and (4) mapping between exported object interfaces and client device controls. We motivate the need for these capabilities by using an example scenario to derive the design requirements for our mobile services architecture. We then present a prototype implementation of elements of the architecture and some example services using it, including controls to audio/visual equipment, extensible mapping, server autoconfiguration, location tracking, and local printer access.

1. Introduction

Researchers have predicted that wireless access coupled with user mobility will soon be the norm rather than the exception, allowing users to roam in a wide variety of geographically distributed environments with seamless connectivity [52]. This *ubiquitous computing* environment is characterized by a number of challenges, each illustrating the need for adaptation. One challenge is the continuously available but varying network connectivity [7], characterized by high handoff rates exacerbated by the demands of spectrum reuse. Another is the variability in client devices: impoverished devices need to push computation into the local infrastructure to allow for application-specific adaptation [15]. A third characteristic is the variability in available services as the environment changes around the client.

It is this third feature that has been least addressed by previous research. This paper investigates novel uses of a ubiquitous network, focusing on variable network services in the face of changing connectivity and heterogeneous devices. We propose that providing an "IP dial-tone" is not enough. We must augment basic IP connectivity with *adaptive network services* that allow users to control and interact with their environment.

In developing this architecture, we have designed, implemented, and deployed in our building the following example services:

- untethered interaction with lights, video and slide projectors, a VCR, an audio receiver, an echo canceller, motorized cameras, video monitors, and A/V routing switchers from a wirelessly connected laptop computer;
- automatic "on-the-move" reconfiguration for use of local DNS/NTP/SMTP servers, HTTP proxies, and RTP/multicast gateways;
- audited local printer access;
- interactive floor maps with a standardized interface for advertising object locations;
- tracking of users and other mobile objects.

In realizing this architecture, we employ a few key techniques:

- augmenting standard mobility beacons with location information, scoping features, and announcements from a service discovery protocol;
- using interface specifications that combine an interface definition language with the semantics of a model-based user interface; and
- hosting scripts in the infrastructure that
  - * map exported object interfaces to client device control interfaces,
  - * compose object interactions, and
  - * automatically remap the destination of object invocations to changing server locations.

The testbed for our experiments [26] includes Intel-based laptop computers with access to a multi-tier overlay network including room-sized diffuse infrared cells (IBM IR), floor-sized wireless LAN cells (AT&T WaveLAN), and a wide-area RF packet radio network (Metricom Richocet). We also leverage facilities in two seminar rooms, a laboratory, and a student office; all contain devices that can be accessed and/or controlled via our software. The physical components of the testbed in one of the seminar rooms (our first prototype, 405 Soda Hall) are illustrated in figure 1.

Our infrastructure builds on and extends the substantial work in mobility support provided by the networking

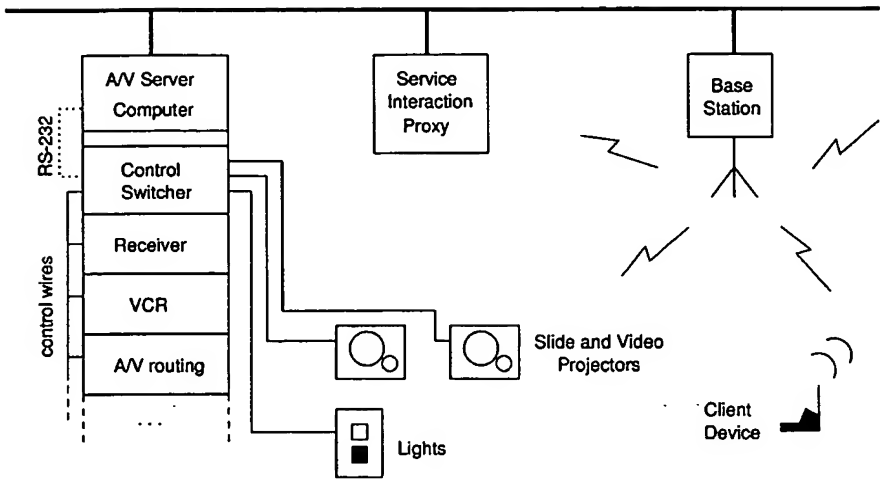

Figure 1. Part of the project operating environment: 405 Soda Hall.

research community. The Mobile-IP working group of the IETF [35], among others, has made great strides in the routing aspects of the problem. Overlay networking [45] has demonstrated the feasibility of seamless handoff between interfaces to different Internet service providers and the reality of ubiquitous connectivity. The developing Service Location Protocol [49], the Domain Name Service [33], and the Session Announcement Protocol [19] address some concerns in resource discovery and management; DNS and the Session Description Protocol [20] address naming. Such efforts have been instrumental in motivating this work and in driving its design.

The rest of this paper is structured as follows. In section 2, we discuss the problems of service provision and provide a framework for describing a service architecture's core functionality. We motivate this with a scenario, deriving a set of functional requirements to be achieved. In section 3, we detail our architecture's prototype implementation and the protocols that allow mobile clients to access the infrastructure. In section 4, we describe the suite of example ad hoc mobile services incorporated into our testbed. In section 5, we unify the design and implementation with some discussion and a layered view of the components. In section 6, we discuss the relevant related work. In section 7, we summarize future and continuing work, and finally in section 8, we present some conclusions.

2. Designing a service interaction architecture

Consider the following scenario:

You are on your way to give an invited lecture.
After parking on campus, you take out your PDA with a wireless network interface. Turning it on, a list of local services is presented. You click on the map service icon, and are presented with a campus-wide map that includes a rough indication of where you are. You select "Computer Science Division" from a list, and a building near you is highlighted. You walk toward it.
As you enter the building, you glance down at your PDA and notice that the list of available services changes. The campus map is now replaced with the building floorplan. Using the new map, you find and enter the lecture hall.
In preparation for your talk, you select "audio/visual equipment" and "lights" from the list of services, causing a user interface for each to appear. Your selections also cause the rooms' equipment to be indicated on the floorplan. You walk to the VCR and insert a tape.
The lecture begins. As you finish the introduction, you dim the lights and start the VCR with taps on your PDA.
At that moment, you realize you have forgotten your lecture notes. Using your personalized printer user interface, you retrieve a file from your home machine and instruct the closest printer to print the file. The printer's location appears on the floorplan.
A minute later, you are notified the print job has completed, retrieve your printout, and return to finish the lecture.

Using this scenario, we imply that users wish to invoke services – such as controlling the lights, printing locally, gaining access to application-specific proxies, or reconfiguring the location of DNS servers – from their mobile devices. But it is difficult to obtain wide-spread agreement on "standard" interfaces and methods for such service invocation, and thus assure the lecturer that his or her device will be interoperable with the advertised capabilities. Thus, the challenge is to develop an *open* service architecture that allows heterogeneous client devices to discover what they can do in a new environment while making minimal assumptions about standard interfaces and control protocols.

Implementing such a service architecture makes it possible to turn client devices into "universal interactors". An *interactor* is, broadly, a device that allows a user to interact with and modify his or her environment. Examples include electronic equipment remote controls and thermostats. A *universal interactor*, on the other hand, is a device that

adapts itself to control *many* devices. It does so by discovering their control interface. A universal interactor thus exploits what it finds in the environment, and varies its abilities as a function of location. It is not a particular hardware component, but instead a way of *using* an existing device.

Realizing universal interactors requires four technical capabilities: (1) device mobility, (2) network-accessible controllable objects, (3) an underlying discovery architecture, and (4) mapping between exported object interfaces and client device control interfaces. These are detailed in the following subsections.

2.1. Device mobility

A critical component of the scenario is device mobility. The client moves from a wide-area network to a local-area network, and between points in the local-area. Basic mobile routing support is an underlying assumption of our architecture. This functionality is available through Mobile IP [35] or similar handoff infrastructure, and can be augmented with network overlays [25] and local multicast [41]. Mobile IP supplies IP-level transparency to changes in location, overlay networking augments this functionality with a policy layer for managing connectivity to multiple available network interfaces (multi-homing), while local multicast provides seamless, low-latency handoff. We build upon this network-layer functionality.

2.2. Controllable objects

Most physical objects provide only manual controls. A *controllable object,* on the other hand, responds to control requests or transmits status information through an exposed interface accessible over the network. This property was illustrated in the scenario when the visiting lecturer controls the VCR and lights from his or her PDA.

To fit into our architecture, it is crucial that objects be augmented with this ability for network-based access. Issues in doing so include aggregation of objects into a controllable unit, addressability/naming, and conflict resolution.

2.2.1. Aggregation

At what granularity must controllable objects be implemented? In keeping with our goal of adaption, we allow it to be arbitrary. Requiring fine-grained controllable objects is difficult because individual objects may be too numerous or the expense of individual control may be too high. For example, while it is possible to make every lightbulb its own controllable object, the sheer number of them in a typical building, the expense of assigning processing to each one, the difficulty of wiring each to the network, etc., would mitigate such a decision. Instead, control functionality could be assigned to a bank of lights, and what is augmented is the switch bank rather than all of the individual lightbulbs. In general, the granularity at which object capabilities are exported should not be specified by the architecture. The difficulty, then, is to allow client controls to aggregate and subset controllable objects components in a manner transparent to the client interfaces above them.

We support this feature by providing clients with a facility for hosting scripts that map exported object interfaces to client device control interfaces and compose object interactions. The scripts leverage the use of interface specification languages for object description, allowing access to subcomponents and composition of remote objects. This disassociates controllable object granularity from actual control granularity.

2.2.2. Naming

Another impact of making controllable objects accessible is that the current infrastructure for naming must be extended to include them. These objects do not have individual IP addresses or session descriptions, but instead are accessible through servers and, due to location-based usage, often have fine geographic scope. Users want to make queries based on geographic information (location), data type (position in a class hierarchy), scope (accessibility range), and the control authority (the "owner" and/or position in an organization hierarchy for dealing with access-control). These properties can dynamically change, and the hierarchies are not strict (i.e., there can be multiple paths from the root). It is unclear whether these four orthogonal components need to coexist in the globally-visible naming scheme (augmenting or acting together as a fully-qualified unique object name), or whether some can be treated as "properties" rather than elements of the name.

Existing methods for naming include using the Domain Name Service (DNS) [33] for objects with unicast IP addresses or the Session Description Protocol (SDP) [20] for lightweight sessions. DNS is a widely-deployed distributed name service. SDP is a container protocol for associating a single name with a collection of application-specific multimedia transports and their (most often multicast) channels. SDP messages are delivered via the Session Announcement Protocol (SAP) [19], an announce/listen protocol that uses scoped constant-bandwidth allocations.

Alternatives for the implementation of object naming include extending DNS with new record types, extending SDP/SAP with new application types and finer scoping, hybridizing the two, or developing a separate hierarchy to match this need rather than overloading DNS and/or SDP/SAP. Further implications of this decision are noted in section 2.3, where we describe service announcement and discovery.

2.2.3. Shared control conflicts

When multiple users attempt to share a controllable object, there is the potential for conflicts in the requests. Existing systems manage this difficulty by providing well-formed application-specific solutions and limiting the set of conflict states. One example is the elevator. Requests to an elevator are not commands, but are instead idempotent inputs to an algorithm that decides an order for actions to take place (if at all). Individual elevators react to input

combinations differently, and this is acceptable. We propose using such application-specific algorithms for controllable objects (encapsulated behind the remote object invocation specification). Leveraging "authentication" features such as locking (minimally coarse-grain, possibly finer) and access levels ("capabilities") can assist in reducing the conflict state set and is very practical; i.e., the "owner" of the device can always override "users", etc.

2.2.4. Cameras as object interfaces

Another approach for interacting with objects is to use video capture augmented with image processing ("computer vision") where applicable. Example uses of this approach include fine-grain object tracking, directionality sensing, and event triggers keyed to particular circumstances [30]. For example, a camera can be used to detect the opening of a door or window. In this case, it is the camera that exports the control interface. Using cameras for such duties has extensive implications for security and privacy control, but is a viable alternative to direct manipulation.

2.3. Service advertisement and discovery

One property of the scenario is that the lecturer was able to discover the existence and location of services while moving. This requires some form of discovery subsystem: a service discovery protocol. The function of a service discovery protocol is to allow for the maintenance of dynamic repositories of service information, advertise the availability of this information, and support attribute-based queries against it.

Service advertisement must scale with both the number of advertised services and the wide-area. Even as larger numbers and classes of devices become network-accessible (e.g., "IP light bulbs"), the bandwidth consumed by these advertisements must scale sub-linearly. Fortunately, though objects can be addressed globally, repository queries may be assumed to show locality, and eventual consistency semantics are acceptable. This allows for the use of a soft-state update approach such as an announce/listen protocol [11]. Thus, these announcements and queries can be scoped and this scoping can provide the basis for hierarchy. Difficulties include that the scoping granularity may be very fine – at the level of individual rooms or network subnets/cells – and that scopes must dynamically adapt to support incremental, independent local deployment of various services.

Our observation is that this requires a technique combining the properties of a distributed name service (e.g., DNS) and an announcement protocol (e.g., SAP). The latter "pushes" unknown names/descriptions to the client to facilitate discovery, while the former allows the client to "pull" information about objects given their names. The hybrid technique is to advertise the location of local name services in addition to object descriptions. This allows a single message to, in effect, advertise a collection of objects, and provides advertisement hierarchy (possibly, but not necessarily, aligned to the naming hierarchy like DNS) with scaling sub-linear in the number of advertised objects.

The Service Location Protocol [49], a resource discovery protocol under development by the IETF Service Location working group, is one proposal for implementing a local-area version of such a service. In SLP, query processing is performed at directory agents and/or in a distributed manner via multicast. SLP is amenable to a wide-area extension leveraging its query grammar, message formats, and security specification.

For basic operation of the system we have running, the only mechanism necessary is a function to allow mobiles to map names to values. These mappings can be set and obtained by querying a local server or via multicast (our current prototype operates via the former.) We describe our own mechanisms for finding the correct local server or multicast addresses and initializing the mappings. Finding one of the correct local servers is similar to delivering the correct SCOPE attribute to the mobile host in SLP. (The SLP SCOPE attribute is used to administratively aggregate an otherwise disparate set of services.)

2.4. Mapping client controls to exported objects

In our scenario, the lecturer's PDA UI widgets remained in familiar locations and in a familiar form even as the devices that the widgets control were changed (e.g., the printer). We now describe some mechanisms for enabling this behavior.

2.4.1. Transduction protocols

A transduction protocol maps a discovered object interface to one that is expected by a given client device. It supports interoperability by adapting the client device's interface to match the controllable object's interface. It allows for custom user interfaces to ad hoc services, such as allowing a virtual "light switch" on a control panel to always control the closest set of lights. Without a mapping function, every change in location might require that a new interface be retrieved.

An issue with transduction protocols is how to map control functions into a UI supported by the portable device. As an example, assume a client device has a two-position switch widget for use with the local light controller. At a visited location, the light controller supports continuous dimming. In this case, the client may substitute a slider widget for the switch. If it cannot do this (or chooses not to), then the purpose of the transduction protocol is to map the on/off settings of the UI to one of the two extremes of the actual dimmer control.

To support interoperability, we allow services to transfer an entire GUI to the client in a language it understands, avoiding the need for transduction. (This is similar to the Java applet usage model but with multiple language support where necessary.) Whenever possible, though, we augment the GUI (or replace the GUI completely) with an interface specification (further described in section 3.7.2). Through

the interface specification, the system discovers the two data types that need transduction. This allow the mapping function to be inferred (heuristically, from a library, or by querying the user) and then installed at the local transducing proxy that sits between the two endpoints. The interface specification can also be used directly to generate a rough GUI when no interface implementation appropriate for the client is available, or when only portions of the controllable objects' interface are of interest to the user (i.e., to conserve screen real estate or to add a button into a user-defined control panel).

The interface descriptions not only allow for data type transducers between client and server, they also provide the critical layer of indirection underneath the user interface. Example uses of this indirection include composing "complex" behaviors and remapping the destination of object invocations to account for mobility.

2.4.2. *Complex behaviors*

Objects have individualized behaviors. We wish to couple and compose these individual behaviors to obtain more complex behaviors within the environment. For example, consider a scenario where music follows you as you move around a building. One behavior of the sound system is to route music to specific speakers. A behavior of location tracking services is to identify where specific objects are located. A "complex" behavior allows us to compose these more primitive behaviors of sound routing and location tracking to obtain the desired effect of "music that follows".

A key problem is that there is no common control interface for individual components. Furthermore, some behaviors may require maintenance of state that is independent of both subcomponents. An example of the latter is instructing the coffee maker to brew only the first time each morning that the office door opens. Another issue is a policy-level difficulty implied by this scenario: resolution of incompatible behaviors. If other users consider music to be noise, the visiting user's music may or may not need to be turned off in their presence, depending on seniority, social convention, explicit heuristics, or otherwise. At a minimum, the system must guarantee that it will detect such incompatibilities and notify the user(s) involved in order to avoid instability (e.g., music pulsing on and off as each individual behavior is interpreted).

Once again, as in transduction, our solution is to use *interface discovery* (learning new objects' input/output data types), paired with the data type transducers (for manipulating those data types) to allow objects to be cascaded to achieve the desired complex behaviors. Additionally, we supply intermediate entities ("proxies") that maintain state that is independent of the constituent subcomponents. This allows for the incorporation of such features as conditional statements and timing information.

3. Implementing service interaction

This section describes some of the implementation details of our architecture.

3.1. *Basic operation*

The prototype allows a mobile host to enter a cell, bootstrap the local resource discovery server location, and acquire and display a list of available services. It also allows users to maintain a database of client-side scripts to be executed when particular services are discovered for use in reconfiguration, local state updates, and to trigger location-dependent actions. Similarly, a set of scripts are maintained in the infrastructure at each site for locale-specific adaption such as transduction and composition. The prototype also allows for simple, incremental addition, deletion, and modification of available local services.

The key components of the compete system are the "service interaction proxy" (SIP), the "service interaction client" (SIC), and the "beaconing daemon" (beacond) programs. These prototypes implement and integrate selected infrastructure components of our overall mobile services architecture. The SIC runs on the client device and provides the base functionality for discovering and managing services. SIPs run at domain-specific granularities and aggregate a group of services with a single set of advertisements. The SIPs also manage the proxies used between client devices and individual services (when necessary). Beaconing daemons run at each base station and are affiliated (not uniquely) with the SIP it is advertising.

An example SIC screenshot is shown in figure 2. SIP and beacond use configuration files and command-line switches, and thus do not have graphical user interfaces.

3.2. *System setup*

Each SIP process maintains a database of the services and service elements that it provides to mobile hosts. An example startup file for such a database is listed in fig-

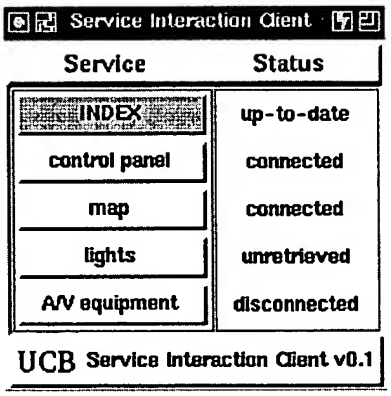

Figure 2. The SIC application GUI is currently a series of buttons that can be used to retrieve and invoke application interfaces.

```
set NAME {
        Soda 405: High-Tech Seminar Room
}
set SERVICES {
        INDEX lights {A/V equipment} map printer {location tracking}
}
set VALUES {
        DNS {128.32.33.24 128.32.33.25}
        NTP {orodruin.cs.berkeley.edu barad-dur.cs.berkeley.edu}
        SMTP {mailspool.cs.berkeley.edu}
        ...
}
set PROPERTIES {
        lights {ISLfile ../helpers/lights.isl version 0.01 \
                location {132 210} appName-tk ../helpers/lights.tk \
                appArchive-tk ../helpers/405/lights405.tar.uue
                appName-tcl ../helpers/lights.tcl \
                appArchive-tcl ../helpers/405/lights405tcl.tar.uue}
        {A/V equipment} {ISLfile ../helpers/htsr.isl location {132 180} \
                version 0.01 appName-tk htsr.tcl \
                appArchive-tk "../helpers/405/HTSR.tar.uue"}
        ...
}
```

Figure 3. An abridged SIP services database example.

ure 3. It contains three types of entries: SERVICES, VALUES, and PROPERTIES. VALUES are used for generic (key, value) lookups. These are useful for, e.g., detecting the need to update server addresses. SERVICES and PROPERTIES are used to specify what, where, and how services are available from that particular location. Each SERVICE has a unique name, and maintains PROPERTIES such as the version number, a pointer to an associated ISL file (described in section 3.7.2), pointers to particular language implementations of user interfaces for the service, and the geographic location (if any) for use with maps. VALUES and PROPERTIES may just be pointers to another SIP, allowing simple incremental deployment to subdomains and yielding a notion of topology.

3.3. Message-level detail

The client enters a cell with a beaconing daemon. The daemon sends periodic broadcasts that contain the bootstrap address and port number of that cell's SIP. The client registers with the base station to establish IP connectivity if it needs to. It then requests the well-known meta-service INDEX, which returns a list of the services available. Based on the contents of the reply, the client renders labelled UI buttons for unknown services, executes scripts in a database to allow for locale-specific reconfiguration, and tells the local SIP to remap the location of running services and set up any necessary widget binding remappings.

When a user wishes to use a particular service, the client software checks its local cache of applications. If an interface supporting the requested application is not there, it asks the SIP for the service's "properties". This is a list of available interface descriptions and/or implementations. It also receives any service metadata (such as version numbers). It then chooses either to download a particular interface implementation (e.g., as a Java applet), or the generic interface description, or both. The SIC then unpacks the received archives, sets up transducers matching the interface description to the device characteristics, and finally executes the GUI.

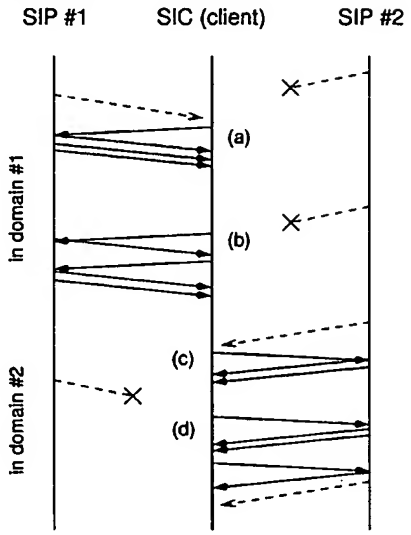

Figure 4. Protocol message timings for a client moving between SIP servers (dashed lines are beacons): (a) INDEX #1 request/reply, (b) request/reply for "lights" ISL file and interface, (c) INDEX #2 request/reply, (d) "lights dim" button press retrieves new ISL file to remap RPC, then completes.

An example exchange of protocol messages for a client moving between SIPs is illustrated in figure 4. Illustrated in this example is a situation where the client retrieves an interface to a light switch in one scope, but then has a later invocation of the service transparently remapped to a new light due to the occurrence of a scope change.

3.4. Client bootstrap

For a client to use services, it must first find the address of the local resource discovery server or the local multicast address where services are advertised. In our architecture, this bootstrap above IP is minimal: there is an indirection

embedded in the mobility beacons. This minimal bootstrap standardizes the mechanism without constraining its interpretation, thereby allowing variation in resource discovery protocols as they evolve.

3.5. Beaconing

Beaconing is required in a system to facilitate notification of mobility-based changes in the relative position of system components. Its use is motivated by inherent availability of physical-level hardware broadcast in many cellular wireless networks and the need to track mobiles to provide connectivity.

Two issues arise once the decision to beacon has been made. The first is which direction to send them: uplink, downlink, or both. The second is what information to put on the beacons, if any at all. (An empty beacon acts as a simple notification of the base station address, available in the packet header.) These issues are discussed in the following subsections.

3.5.1. Beaconing direction

In terms of choosing whether to have client devices or infrastructure servers beacon, existing systems can be found which have made either choice. Client beaconing is used in both the Active Badge [21] and PARCTAB systems [37], while server beaconing is used in Columbia Mobile IP [22]. IETF Mobile IP [35] utilizes both periodic advertisements *and* periodic solicitations.

One might expect that the different policies optimize for different applications' operating modes. This is indeed the case: there are trade-offs in such a decision, as it varies allowances for privacy, anonymity, particular protocols' performance, and scalability.

There are a number of metrics for considering qualitative trade-offs between the two decisions:

- **Power:** Less power is consumed at the mobile by periodically listening than by periodically transmitting, but this difference can be mitigated by hardware/MAC design [44].
- **Detection:** When base stations (BSs) beacon, mobiles need not transmit to detect when all contact is lost. When clients beacon, BSs need not transmit to detect user mobility.
- **Multiples:** With BS beaconing, detection of multiple beacons can be used to assist handoff. With client beaconing, the number of received beacons specifies the number of clients in the cell.
- **Location anonymity:** When BSs beacon, anonymity is preserved for non-transmitting mobiles; when clients beacon, the granularity of the infrastructure is invisible to users.
- **Geographic mapping:** BS beaconing maintains a consistent mapping between geography and beacon broadcast cell; client beaconing maintains a mapping of clients to multiple cells.
- **Bandwidth scaling:** BS beaconing implies less beacon traffic per cell given a natural many-to-one mapping of mobile hosts to base station cells. Conversely, client beaconing optimizes for very small overlapping cells. (Assuming other parameters remain constant.)

Our system uses base station beaconing. We believe this is the correct design choice for three key reasons: the support for user (rather than infrastructure) anonymity, better bandwidth scalability in a network where there are many MHs per BS, and because power is more precious on mobile devices.

3.5.2. Beacon augmentation

The second question is whether to augment mobility beacons with additional data. Augmenting beacons with application-specific data does two things. It makes data available to mobiles *before* registration (in the Mobile IP sense), allowing the possibility of "anonymous" access to this broadcast data (at a cost of management overhead and increased beacon size due to the piggybacking). It also aligns the mobility beacons with a form of announce/listen protocol that has limited announcement timer adaptability. The announcement timer can only be set to discrete multiples of the base beaconing rate (where the base beaconing rate is the rate determined to be sufficient to detect handoff within some acceptable latency.)

Possible uses for such piggybacked beacon data include:

- merging of other periodic broadcasts to amortize header and MAC overhead (e.g., NTP beacons [32], Mobile IP foreign agent advertisements);
- pricing information useful to the host to determine whether to register;
- any form of commonly accessed time-variant data;
- a list of some or all of the available services in the cell;
- "tickets" for providing scoped access control (discussed in section 3.6).

The utility of beacon payload augmentation is highly dependent on the direction of the beaconing, traffic patterns, and application mix. An argument against augmenting beacons at all is that orthogonal applications should not mix their data units that may have been "properly" sized by the application (cf. application-level framing [12] or joint source-channel coding [31]).

We choose to augment our beacons with bootstrap information, a ticket for scoping of services, and a dynamically configurable application-specific payload. The encoding is shown in figure 5. One common application-specific payload is the contents of the cells' INDEX, allowing for anonymous assessment of available services and a reduction in service discovery latency.

Whether merging data into beacons is a benefit depends on the metric of evaluation; our choices are only heuristics. We are still trying to quantitatively determine which data, if any, is best dedicated to these bits for optimizing reasonable client-driven workloads.

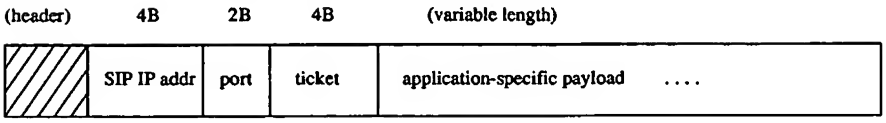

Figure 5. The service beacon encoding includes bits for the service interaction bootstrap and location queries. Not shown are the details of any particular mix of application-specific payloads.

3.6. Scoped access control

Making services available to visitors brings up a host of general security issues, including those specific to the wireless domain [5,9,16]. In addition to the need for standard cryptography-based security at the granularity of administrative domains, service interaction systems require localized access control: the expectation that environmental changes can only be affected by people in that environment (e.g., lights cannot be turned off by a person across the country) has been broken. Maintaining this norm is important when extending existing human social metaphors into an environment with controllable objects.

We address this by embedding *tickets,* random fixed-length bit vectors, in the mobility beacons and requiring that current ticket be included in all communications to servers. Periodically changing the tickets in an unpredictable way (randomly) and scoping the broadcast (implicitly via the cellular wireless network broadcast cell[1] or explicitly with IP multicast scoping) prevents remote access from nodes *even on the access control list* that are not local. This pairs the geographic scoping of the environmental controls (what we cannot control) to the topological scope (what we can control). This ticket-based exclusion can be overridden by having a local agent separately multicast or unicast the ticket when necessary, but by making access topologically restricted by default, we better emulate the existing paradigm.

3.7. Client interfaces

3.7.1. Motivating interface specifications

Clients can be computationally impoverished, have variations in display (color depth, resolution, screen size), support different interface paradigms (keyboard, pen, touch), and are often not preconfigured (allowing them to be interchangeable).

Due to the need to support such end devices, especially extremely resource-poor PDAs, our architecture focuses on providing thin client interfaces and offloading processing to proxies. Recognizing that expecting custom UIs to be available for all services on all different types of hardware is not realistic, we propose exposing controllable objects through an *interface specification language* (ISL). The ISL exposes the syntax of each service's control interface. The ISL is used *in addition* to implementations of an interface in various languages, thus allowing for compatible but independent coexisting interfaces. Upon discovery of a service, the client device checks to see if a language implementation is available that it can support, and if not, it uses the ISL file to learn the syntax of RPC calls and call parameters that can be used to access the service. It additionally allows the device to adapt the representation to a format appropriate for the device's characteristics (possibly transduced at a proxy), and allows the user to place the elements manually and independently of one another for fine-grain control. Automatic UI layout heuristics can also be used on an ISL file directly.

3.7.2. Interface specification

The conventional notion of an Interface Definition Language (IDL) (e.g., the CORBA IDL [34]) is to specify method names, parameters, parameter data types, and parameter-passing conventions for remote object invocation. The basic function of a Model-based User Interface [46] is to specify interfaces as structured widget hierarchies along with sets of constraints. The actual interface is derived from the model at run time or through a compilation step. This allows interfaces to be amenable to arbitrary adding and deleting of elements. Our goal is to unify these approaches, allowing remote object API descriptions to be augmented with UI models. This hybrid would allow both composition of the object invocations (RPCs), and separate subsetting/aggregation of object UI elements. We call this combination an interface specification, and its grammar an *Interface Specification Language,* or ISL.[2]

As concrete examples, we would like to allow an application to buffer DNS queries (calls to one object method) through a local cache (calls to another object), while instantiating the new functionality through the original UI. Similarly, an RPC spawning an audio conference can be rerouted to a client script that first reduces the volume of the room's music player and *then* passes along the original RPC. In both cases, if the individual components were built as monolithic "applications" without an accompanying ISL, there would be no exposed indirection allowing us to perform the remappings below the user interface. The use of an ISL explicitly exposes this layer of indirection, allowing transparent remapping of individual interface elements. This indirection is critical for allowing services to be composed.

An important design criteria in such a IDL/model-based-

[1] RF localization is at best approximate due to effects such as multipath. IR may be a preferable transmission media for tightly controlled localization.

[2] The ISL, though called a 'language', can be implemented atop another high-level language syntax or document format, such as in Java or as an XML schema [14].

UI hybrid is that it be robust enough to be applicable to various classes of client devices (each with their own implicit assumptions about widget implementation), yet simple enough to be manageable in terms of syntax parsing and authoring difficulty. Our current implementation has interfaces manually implemented in Tcl/Tk and an ISL built atop an ad hoc grammar tuned to Tk; we are moving to XML for the ISL.

An example of a very similar need but in a different domain is the Universal Remote Console Communication Protocol [48]. This work allows different interfaces to be made available to disabled users by requiring applications to expose their UI as a well-defined interface. As in our case, it is the exposed indirection that is critical to enabling the work.

3.7.3. Prefetching

As an optimization, clients can prefetch the ISL files for active services. We illustrate with a concrete example from our prototype. As the user moves between rooms, the light controller application UI remains the same. When the user changes the lighting in a previously unvisited cell, the client application must first send the new SIP a request for the lights ISL file. This enables the RPC command invoked by the existing interface to be remapped to the new lighting server. (This particular message exchange was illustrated in figure 4.) Such late-binding is used to conserve bandwidth on the wireless link; the total number of ISL files may be large and the client may use one only infrequently.

The problem with late-binding is that the entire operation latency is seen by the end user; in practice it can be perceived as a possible error condition. (The button "does not work" for a number of seconds after it is invoked, and for this period it should probably be greyed out in the UI.) This delay can be minimized by transparently remapping the interface elements to the new server as soon as possible. To do so, we add one bit of per-service state, "active vs. inactive". This flag is set to "active" whenever there is an RPC call to that service, and reset to "inactive" by a timeout. Upon receipt of any beacons with a new SIP, services with the "active" bit set (and available in the new location) have their new ISL files prefetched automatically, thereby reducing their invocation latency. (Additionally, the INDEX meta-service is always prefetched in our current implementation.) Delays can be further minimized through mobility prediction [28], allowing prefetching in response to assumptions about user mobility patterns.

Prefetching is important in this domain because the delay experienced by an end user using a high-latency, low bandwidth wireless interface can be substantially mitigated through the use of prefetching rather than demand-paging.

3.7.4. Client-side security

Client interfaces may be passed to the user from a visited infrastructure in a number of forms. Because in most cases we are only transferring interface code rather than fully general "mobile code", it is probable that a sandboxed environment (such as with Java applets, Safe-Tcl, or Janus [17]) can be used without constraining the service's functionality. This is another benefit of the proxy-based access model: it segments the security domain, thereby screening more of the system internals from the user. This approach also aligns well to the restricted Java applet communication security model, where messages can only be sent to the applet provider (the SIP in our case).

3.8. Naming scheme

We express controllable object names as a globally-unique fully-qualified object name (FQON) and a tuple of properties. Some properties are required while others are optional but standard. (Service-specific properties are also allowed, of course.) The set of basic properties includes:

- geographic location as a name hierarchy of maps that contain the object,
- the data type of the object as a class in a class hierarchy and a version number,
- the name of the "owner" of the object to indicate where and how to obtain authenticated credentials (if necessary),
- a pointer to the controlling server process (machine/port).

A common additional property is a set of pointers to "peer" controllable objects and tags that note the I/O data flow that makes them peers. These are useful for specifying, for example, that the (analog) input source to a monitor is a particular A/V switch.

We have not as of yet incorporated scope information into the object names/properties, but this remains as a critical area for future research.

4. Prototype mobile services

In addition to the prototype service discovery and interaction implementation, we have developed a number of services using the framework. These include maps that specify discovered objects' positions, autoconfiguration, location tracking with privacy allowances, audited printer access, and interfaces to audio/visual equipment. We now describe each in turn.

4.1. Maps

Given a widely distributed service interaction system supporting very fine-grained services, management of even the subset of information available to the client becomes non-trivial.

We have experimented with using maps for explicit management of these services at multiple locations. Map content can be separated into three domains: network connectivity (topology and link characteristics), physical geography (object locations and floorplans), and administrative

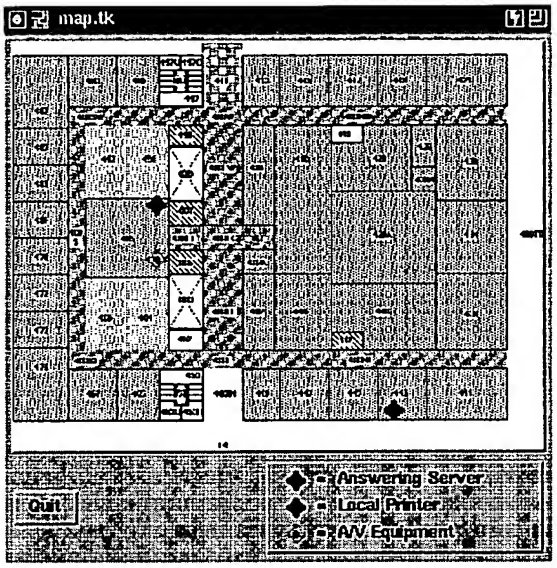

Figure 6. Map with discovered object locations, configured for a user in the RF cell including room 405. Clicking on entries spawns the interface to that entity.

domain (access rights, pricing, hierarchy). Our prototype – of which an example view is shown in figure 6 – focuses on physical geography: allowing objects to note their location on multiple overlapping maps and to receive requests passed through the map interface via a button press on the service indicator on the map. Map positioning in the prototype is based on using relative coordinates. It is designed to allow objects to position themselves without knowing exactly which map(s) the user is viewing, and allows maps to maintain hierarchical ("containment") relationships in a distributed, extensible manner.

It works as follows: services express their location relative to some map by holding a pointer to the map itself and specifying a bounding box or point on the map. To provide the hierarchical containment, maps may themselves have pointers to other maps, and similarly indicate their relative location and size in the "neighbor" maps just as services do. In this way, (coarse-grained) positioning of disparate services on disparate maps may be maintained without requiring any absolute positioning information (i.e., requiring that everything use GPS coordinates) – represented objects and locations can be relocated to new maps where a chain of containment can be found, and can be sized in accordance with the scale of the map(s) they are directly located on. The map hierarchy thereby maintains the same characteristics as a SIP hierarchy: it can be arbitrarily nested and extended to subdomains without affecting other maps or requiring objects to relocate themselves.

This technique can be extended for use with an absolute-positioning-based notation for physical geography, as in the proposed "LOC" DNS record type [13] extended to objects without IP addresses. In a system with absolute positioning, neighbor pointers are unnecessary; the relative-positioning-based approach, though, is more robust to precision errors and usable when no reference absolute positions are available.

This very simple approach sits in stark contrast to the various complex transform mappings used in Geographic Information Systems. Though there is an obvious fundamental limitation in accuracy and consistency a relative approach can achieve, the novelty is the simplicity – it is easy for a non-expert to add a new map to a collection in any form, ignoring difficulties like rotation orientation ("which way is North?") and maintaining object constancy ("are these two roads the same?").

4.2. Proxy and gateway autoconfiguration

There are a number of useful local intermediate agents ("gateways" or "proxies") that could be made available to visiting users. Examples include web proxies that perform on-demand dynamic transcoding [15], WWW data caches, real-time media transcoders [3], and multicast-to-unicast gateways for multicast-unaware client devices (i.e., most PDAs).

Additionally, though, proxy/gateway/server autoconfiguration is important in a mobile environment for more than just efficiency. Using the "best current practice" technique of hard-coding DNS servers as `/etc/resolv.conf` entries or in the Windows registry, if a user were to move from a location behind a firewall to one that is not, all lookups will fail until an out-of-band technique is used to find a new server and manually update the entry. The Network Time Protocol is dependent on server location due to its use of RTT estimation, and is therefore especially suitable for use with automatic reconfiguration. A failure to keep accurate time can break some security systems, notably Kerberos. Spawning a local RTP gateway/transcoder for unlayered data in MBone sessions may be necessary if movement has changed the bottleneck link to sources or to facilitate local management of inter-session bandwidth sharing [1].

Autoconfiguration also adds a level of fault tolerance. If a network link goes down, SIP beacons coming across the failed link will stop. The client can wait for other beacons to be obtained (cf. in an overlay network with vertical handoff [45]), and reconfiguration to the new servers will happen transparently.

Our current implementation simply allows for callbacks to be set that track VALUE entries in the SIP databases. A more complete and detailed solution to this problem applied to media gateways is described in [2].

4.3. Location tracking

Location tracking has been addressed in other systems [21,39], and can be implemented by allowing queries against foreign agent registrations or routing table entries (as appropriate). These systems suffer from the limitation that client devices must be turned off or not carried to ensure users are not vulnerable to continuous detection (e.g.,

while in a restroom). Additional effort can be expended to mitigate this limitation, such as by having queries sent to a group of cells rather than a single cell [43], but a complete solution ensuring security must avoid a traffic analysis attack even when the infrastructure is trusted.

In our system, devices may be carried continuously while still allowing for detection avoidance. This is due to the fact that clients do not periodically transmit (as described in section 3.5).

4.4. Printer access

One of the most common examples in the resource discovery literature (and commonly requested end user service) is local printer access. In our implementation, after the discovery protocol finds the local printer and notes it on the map, clicking on it (or on the "print" SIC button) pops up a dialog box that can be used to send a postscript file on the client to the local print server. The server then checks the data, logs the request, prints the file, and returns any status and/or error messages.

4.5. Motorized cameras

We have built software to control both the Sony EVI-D30 and Canon VCC1 motorized cameras. A unified user interface for the four cameras in 326 Soda is shown in figure 7. Operations supported include pan, tilt, zoom, speed control, and setting of software presets.

The camera controls were used extensively by members of a UCB class ("CSCW using CSCW") with remote participants in the Fall of 1997. The controls provide better remote monitoring of whoever is speaking and allow for framing a group of speakers into a single view.

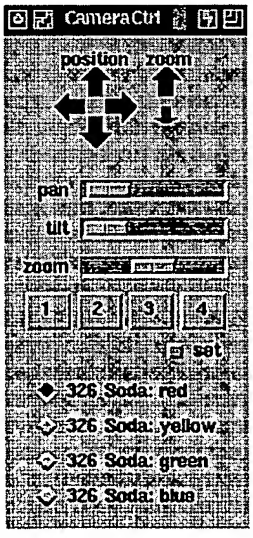

Figure 7. Screenshot of the user interface to the camera controls.

4.6. 405 Soda Room interaction

The "high-tech seminar room", where weekly MBone broadcasts of the Berkeley Multimedia and Graphics Seminar take place, is equipped with a variety of equipment: two slide projectors, a light controller, a video projector, a VCR, a receiver, a DEC workstation, and an Intel PC. All the devices are attached to an AMX corporation control switcher or routed via an AMX router. The DEC workstation talks to the AMX via a RS-232 serial connection, which allows the workstation to act as the control interface.

In 1993, Bukowski and Downs designed a library for accessing the AMX from a workstation for use in a similar room [10]. They also produced a client/server package utilizing the library. We leverage their work, along with a new version of Tcl-DP [42] as the RPC interface, and extend it for use in our environment.

4.6.1. Design and architecture

The application, like others in our architecture, is built using the principle of *application partitioning* [51]. Due to the potential lightweight nature of clients, the server is required to bear the brunt of the effort to support fault tolerance, access control, and other such duties. Features can be added to make the internal system interactions more robust with little or no change to the client-side code.

The server runs on an extended Tcl/Tk wish shell which includes the base AMX functions. The server opens an RPC socket and listens for requests to convert to AMX commands. It is also responsible for maintaining the hard state of the system. This leaves the clients free to act as only a UI and cache for soft state.

All communication with the AMX is done through an intuitive scripting language (e.g., `pushButton vproj power-on`). As an example, if the client executes a command such as `pushButton receiver power`, the routines of `AMXHelperLib.tcl` translate it into a remote procedure call for `send-AMX-command 1 3 75`. The request is forwarded through the wireless base station to the server application. The server, in turn, maps this request into a packet for the AMX, which it sends down the RS-232 serial link. Upon receipt of the message, the AMX's embedded controller routes the signal to its third slot (the one for the receiver) instructing it to to turn on channel 75 (power). This causes the correct stored IR frequency to be broadcast down a wire connected to the standard remote control IR port on the receiver. Other wires use mechanical calipers for manipulation of the slide projectors and variations in voltage levels to adjust the lights.

4.6.2. Room interfaces

Our initial implementation of the room controls includes two separate monolithic Tcl/Tk programs for the room's control, one a superset of the other. The first handles only the lights, while the second handles the most useful buttons (showing them all is excessively complex). The latter interface is shown in figure 8.

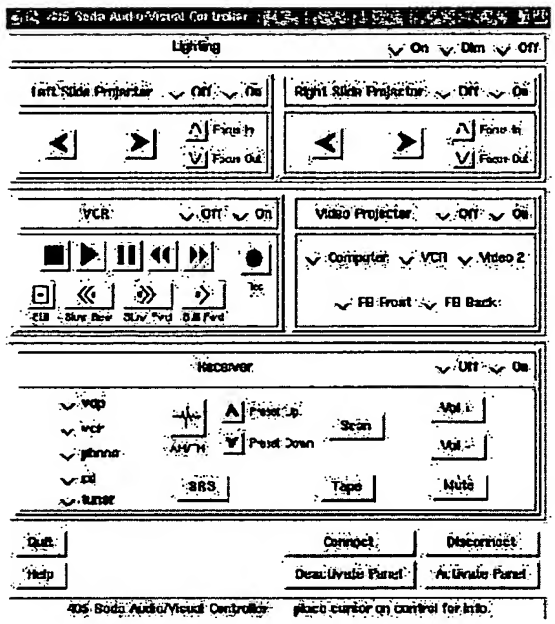

Figure 8. Screenshot of the monolithic user interface to the A/V equipment.

It was these implementations that led us to observe the utility of functional inclusion and the need for variability in the interfaces. It also led us to realize that independent objects should be composable. With such a design, users could create unique UIs that makes the most sense for themselves by leveraging the scripting language and ISL. We are working on allowing the user to maintain sets of service elements by manipulating the interface specifications transparently, for example by dragging-and-dropping individual elements to and from a toolbar.

4.6.3. State management

Ideally, requests to the AMX could be idempotent, and no state would have to be maintained in the system. However, by the nature of the equipment to which it is attached, AMX requests are *not* idempotent, and cannot be coerced into idempotent versions. For example, if we want to turn on the receiver, the only request we can give is equivalent to "toggle receiver power", which may very well turn the receiver off. The only way to know the effect of this request beforehand is if long-lived state variables are maintained.

Because bandwidth between the client and server is often valuable, state variable updates need to be minimized. Our server is responsible for maintaining consistent state between and during client sessions. Clients are responsible for querying the server for the state info upon connection initiation. For consistency, clients are required to send an update request to the server to ask for state changes. Clients may only commit the change upon receipt of an acknowledgment.

Another issue is dealing with inconsistencies due to manual events. Devices are unable to inform the AMX when a user presses a button on their front-panel. If a user wants to insert a video cassette into the VCR, he or she must first turn it on; the AMX does not register this manual event. Since this is such a common problem, we accept that the state will, at times, become corrupt. We equip the user to correct such inconsistencies via the two buttons at the bottom of the client interface labelled "Deactivate Panel" and "Activate Panel". Whenever a discrepancy in the state occurs, the user can deactivate the panel. All the state-related buttons will then only modify the state variables (on both the client *and* the server) and *not* make requests to the AMX. This means the user can reconcile the information on the panel with reality, then reactivate the panel and once again right the system to a consistent state.

5. Discussion

Section 2 described the design issues and problem characteristics of service provision. Sections 3 and 4 gave details of a prototype implementation. We now attempt to better unify and situate these views.

A layered view of our architecture is presented in figure 9. This representation exposes how the various mechanisms described in this paper interrelate. It also illustrates how alternative mechanisms could replace the particular ones we have chosen without affecting the overall service architecture. For example, if some to-be-determined Service Location Protocol SCOPE delivery mechanism were to replace our augmented beaconing mechanisms for location management, the interface discovery and data type transduction could remain unaffected.

The lowest layer is the *Announcement* layer. It lies directly above the network and transport layers and implements the basic service bootstrap mechanisms. This includes the embedding of local server information in the

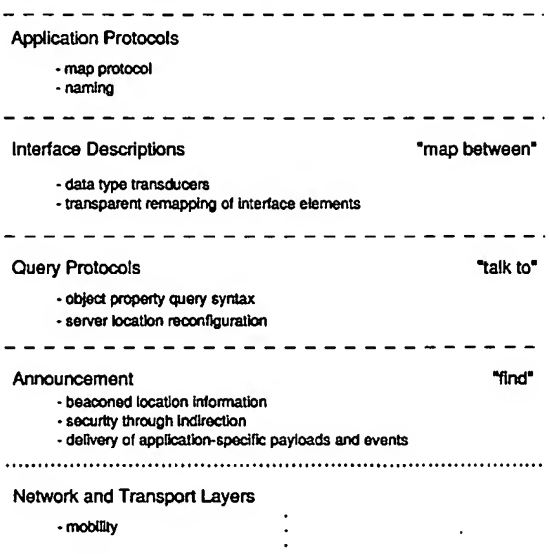

Figure 9. A layered view of the mobile services architecture.

beacon payload, the ability to implement scoping mechanisms through the indirection gained by the beacons, and the possibility for additional application-specific payload augmentation as a performance enhancement. Its most basic function is to *find* servers and users.

The *Query* layer uses server location information from the announcement layer. It adds the ability to interact with found entities such as resource servers and service interaction proxies, and provides a structure for attribute queries (e.g., requests for a device-specific service applet interface). Its most basic function is to allow entities to *talk to* other entities.

The *Interface Descriptions* layer is built on whatever query protocol is exposed by the query protocol layer. It defines the set of possible interface descriptions and their semantics. Its most basic functionality is to *map between* the client device interface and the interface advertised by discovered objects.

The highest layer is the *Application Protocols* layer. It uses the interface description language exposed by the interface description layer. It encapsulates application-specific state or data not captured by the lower layers. Examples include attaching semantic meaning to particular names (i.e., "camera", "power") and defining relationships between data values (i.e., the map hierarchy).

Though we have focused on access for wireless mobile clients, facets of our approach are applicable to wired networking. Examples include automatic reconfiguration of server location (for fault tolerance) and scoping of access to services through limited broadcast of tickets.

6. Related work

The Rover [23] and Wit [51] systems also recognized the need to split applications into a lightweight front-end and more heavyweight proxy at the last hop wireless link. Rover allows the pieces that comprise the partitioned whole to migrate between these two points, but the implemented prototype applications generally only exploit this for moving application data units such as mail messages, news articles, web pages, or calendar entries.

The Service Location Protocol (SLP) [49] is an example resource discovery and service registration mechanism that can also function as a fine-grained name service. Open issues include its lack of an explicit scope hierarchy and peering equivalent to our use of pointers in a service database. Our mechanism for dynamically updating the current SIP location could be adapted as a scope discovery mechanism and coexist with other such mechanisms in the proposal (i.e. having a SCOPE DHCP option).

The seminal PARCTAB [37] and Active Badge [21] systems, along with related work by Schilit [38–40], were among the first to attack the issues of client applications and network support for mobility in tandem. We borrow much from this work, including the focus on mapping, event notification, and support for impoverished devices. There are some key differences. We support distributed servers, rather than a centralized repository. We employ discovery mechanisms, interface code mobility, and generalize to heterogeneous devices; these are unnecessary in their local-area, homogeneous environment with pre-installed custom applications. We use server beaconing rather than client beaconing, and allow the beacons to bootstrap resource location, define scope, assist fault detection, and provide for some location management.

A transportable X display [53] is a variation on interface code mobility; it moves users' *existing* interfaces as they move, not unknown applications' interfaces or interface descriptions. It has the advantage that applications need not change at all, but suffers from the limitations that (1) it does not support transformations of the interface to formats more suitable to particular client devices, and (2) it does not expose a layer of indirection underneath widget invocations.

The Mobisaic [50] and Dynamic Documents [24] projects support a HTML-based structure for varying, location-dependent interfaces. Our scheme generalizes these approaches by incorporating resource discovery and aggregating/subsetting different interface elements.

The Georgia Tech CyberGuide project [29] focuses on prototyping applications augmented with various positioning systems, potentially without communications at all. Using such an approach requires that devices be manually adapted to new environments.

Our conception of a "proxy server" is based on the model expressed explicitly in the Berkeley Client/Proxy/Server model [15] and implicitly in other work [3,8] that places application-level or network-level entities near, but not at, the endpoints of communications. This is another way of thinking about Active Networks [47], driven by end-to-end design principles [36]: push agents as close to the endpoints as possible, but no further. This concept of leveraging the well-connected, computationally powerful side of the wireless link (via "proxies" or "agents") pervades mobility research. It is also driven by the growing availability of workstation farms [4] designed to provide compute resources for just such applications.

7. Continuing work and future directions

Our continuing work involves iterating over the design, refining the implementation, and investigating various other approaches.

7.1. Wide-area issues

The current implementation has been tested only in a local area environment; work is continuing as to the specifics of how such servers aggregate (with union and intersection operations) and their hierarchy. This relates to naming issues and query semantics.

7.2. Building control and support

We are working with building architects and engineers at the Center for the Built Environment[3] to incorporate devices such as the centralized heating and air conditioning, vents, fans, and temperature/humidity sensors into our system. This could allow users to close the environmental control loop and adapt areas in accord with user preferences as they move. Additionally, we hope to apply this model to the corporate environment team-based work process, allowing per-user location-based interfaces. For example, a R&D person visiting the Accounting Division is probably interested in different services than the local workers in the same area.

7.3. Delegating operations

In general, mobiles may be allowed controlled access to CPU resources directly rather than configured services. This allows custom installation of "last-hop" network protocols, codecs, and security modules that are too compute-intensive to run on the end-client (e.g., for allowing the use of end-to-end session keys in an untrusted domain, for delta-encoding data, or for deploying private handoff prediction.) This requires a management layer for implementing policy decisions granting access to bandwidth, disk, and CPU. It also requires a mechanism for securely delegating operations [27].

7.4. Queued RPC

Queued RPC mechanisms [6,23] support disconnection and link variability by incorporating application-managed messaging state. Queued RPC and asynchronous notification support is not incorporated into our system, but could be. On the other hand, applications should also be able to ignore failed RPCs rather than queuing them, a more appropriate paradigm for situations such as with equipment interaction – the client interface is designed to express the current state of external processes and messages can specify idempotent operations.

7.5. Maps

We wish to add additional functionality to our map application, including the ability to tie together physical geography to network connectivity. Servers could be pinned to their location on the floorplan and the connectivity graph automatically overlayed as it is discovered. Also to be added are the specifics of the administrative domains: overlaying the list of services available at groups of servers on the map, and extensions to illustrate hierarchy.

7.6. Interface specification grammar and compiler

A full specification of the ISL grammar and UI generation for different platforms is work-in-progress. Candidate base languages include HTML, XML, Java, the CORBA IDL, or a hybrid.

[3] http://www.ced.berkeley.edu/cedr/cbe.

7.7. Geographic locality

Currently there is no notion of requiring the tie between physical geography and network topology to be explicit. Users given IP access are expected to navigate through the global Internet where little or no locality is exposed even though it can be exploited. For example, the only hints of geographic information are out-of-band channels, heuristically through the IP interface domain name ("whitehouse.gov" in Washington, DC), IP address-to-city registration mappings available through the WHOIS service, or possibly the experimental DNS "LOC" record type [13].

Work has been done to allow clients to recreate these topological relationships for a small class of services using limited support from the network [18]. We would like to overload our hierarchical service infrastructure with this functionality directly. To do so, each service interaction server maintains "pointers" to others in the hierarchy and to peers. The pointers are then links in a geographic chain similar to the more familiar concept chains used in the WWW. In other words, just as HTML hyperlinks associate data based on content without regard to geography, neighbor links associate network topological locality without regard to content. Such links can be set up either manually with multi-lateral peering agreements (people agree to link topological neighbors), through occasional multicast expanding ring searches, or by inferring neighbors through the name and scope embedded in service advertisements. This requires no router support and can be incrementally deployed. As more links maintain a service advertisement beacon with these pointers, the leaves of the hierarchy would be filled out, allowing clients to infer a view of the geographic structure from their location. This gives us the possibility to enable a form of "window shopping" on the Internet, where neighbor networks can be queried to see what is available "next door".

7.8. Multimedia collaboration control architecture

The current suite of multimedia collaboration tools (vic, vat, wb, etc.) is focused on use at the desktop, with local control of each application through its user interface. In other words, the participant is also expected to be the controller.

These applications are now finding use in less traditional environments. One concrete example of this is the MASH Collaboration Laboratory, where media streams are sourced and sinked from a large number of non-computer devices. These devices (cameras, TV monitors, A/V routing controls, etc.) require remote (distributed) control to allow for the development of aggregate control applications that can configure such devices in combination.

We are developing a control infrastructure that can support such applications and developing prototypes for us-

ability studies. The hope is to provide users with robust, intuitive room controls rather than requiring an attending technician to take care of such details. Additionally, the distributed control infrastructure will provide the mechanisms through which remote participants' applications can be controlled out-of-band (modulo policy-level access controls). Such mechanisms would relieve the need for users to receive control instructions (i.e., "Please turn down your source volume") from technicians or advanced users through a sideband (or worse, in-band) channel.

7.9. Conference control primitives for lightweight sessions

The goal is to design a set of control mechanisms from which a wide variety of conference control (e.g., floor control) policies can be built. The base component is an announce/listen protocol to support "voting", much like SCUBA sender-interest messages or RTCP receiver reports. Atop this message-passing framework are mechanisms for specifying the style of shared control (i.e., how votes are tabulated) for each element in the collaboration session. Also, we envision incorporating standard strong-crypto solutions for authentication and encryption to support access control lists and for assigning participants to "ownership classes" for the various objects in the environment.

A related open issue we are exploring is whether individual receivers and application-specific gateways should unify disparate announce/listen protocol messages. It seems that the "global, constant-bandwidth" allocations used by these protocols (i.e, SAP, RTCP, SCUBA) should not simply be summed as new protocols are deployed (a difficult predicament for low-bandwidth networks), but could instead share a single allocation. The hope would be to reduce the required announcement bandwidth by avoiding repetition of redundant data and to reduce consensus latencies by allowing individual protocols to adapt their share of a static allocation as necessary.

7.10. Anonymity versus community

In a shared environment like a meeting room, it is obvious when someone gets up to adjust the lights. In such an environment, it might be more appropriate to announce the controlling actions to everyone rather than keep the action anonymous. Such announcements would leverage social convention to ensure that users inside the protection domain still manually limit their actions.

Understanding how human interactions and the sense of community change given remote access to shared services requires further study.

8. Conclusions

We propose that providing an "IP dial-tone" is not enough. We must augment basic IP connectivity with *adaptive network services* that allow users to control and interact with their environment. The challenge is developing an open service architecture that allows heterogeneous client devices to discover what they can do in a new environment, and yet which makes minimal assumptions about standard interfaces and control protocols.

We then present an architecture for "universal interaction", allowing a device to adapt its functionality to exploit services it discovers as it moves into a new environment. Four capabilities are needed for a comprehensive solution to this problem: (1) allowing device mobility, (2) augmenting controllable objects to make them network-accessible, (3) building an underlying discovery architecture, and (4) mapping between exported object interfaces and client device controls.

We employ a few key techniques to realize our architecture:

- augmenting standard mobility beacons with location information, scoping features, and announcements from a service discovery protocol;
- using interface specifications that combine an interface definition language with the semantics of a model-based user interface; and
- hosting scripts in the infrastructure that
  - map exported object interfaces to client device control interfaces,
  - compose object interactions, and
  - automatically remap the destination of object invocations to changing server locations.

We also provide a detailed description of our prototype implementation of the architecture and a number of example services in use at the UC Berkeley CS building.

Acknowledgements

Thanks to everyone in the BARWAN, Daedalus, GloMop, and MASH projects – especially Hari Balakrishnan, Venkat Padmanabhan, Elan Amir, and Mark Stemm – for assisting our efforts, providing tools, and help in maintaining the environment.

Edouard Servan-Schreiber and Larry Rowe provided help with initial phases of this effort. Richard Bukowski and Laura Downs provided the base AMX control software. James Landay, Mark Newman, and Steve McCanne supplied ideas for extending this project, feedback, and discussion. McCanne, Rowe, and Stemm also provided detailed comments on an earlier draft of this paper.

This work was supported by DARPA contract DAAB07-95-C-D154, DARPA contract N66001-96-C-8508, NSF infrastructure grant CDA 94-01156, the California State MICRO Program, Hughes, Metricom, Daimler-Benz, PCSI, and GTE.

References

[1] E. Amir, S. McCanne and R. Katz, Receiver-driven bandwidth allocation for lightweight sessions, in: *Proc. ACM Multimedia '97* (November 1997).

[2] A. Amir, S. McCanne and R. Katz, An active service framework and its application to teal-time multimedia transcoding, in: *Proc. ACM SIGCOMM '98* (August 1998).

[3] E. Amir, S. McCanne and H. Zhang, An application-level video gateway, in: *Proc. ACM Multimedia '95* (November 1995) 511–522.

[4] T. Anderson, D. Patterson, D. Culler and the NOW Team, A case for networks of workstations: NOW, IEEE Micro (February 1995).

[5] A. Aziz and W. Diffie, Privacy and authentication for wireless local area networks, IEEE Personal Communications (First Quarter 1994) 25–31.

[6] A. Bakre and B. Badrinath, Reworking the RPC paradigm for mobile clients, Mobile Networks and Applications 1(4) (January 1997) 371–386.

[7] H. Balakrishnan, V. Padmanabhan, S. Seshan and R. Katz, A comparison of mechanisms for improving TCP performance over wireless links, IEEE/ACM Transactions on Networking (December 1997).

[8] H. Balakrishnan, S. Seshan, E. Amir and R. Katz, Improving TCP/IP performance over wireless networks, in: *Proc. 1st ACM Conf. on Mobile Computing and Networking*, San Franscisco, CA (November 1995) pp. 2–11.

[9] D. Brown, Techniques for privacy and authentication in personal communications systems, IEEE Personal Communications (August 1995) 6–10.

[10] R. Bukowski and L. Downs, User's guide to the 608-7 computer control system, December 1993, UC Berkeley CS260 class project.

[11] D. Clark, The design philosophy of the DARPA Internet protocols, in: *Proceedings of SIGCOMM '88* (August 1988) pp. 195–206.

[12] D. Clark and D. Tennenhouse, Architectural considerations for a new generation of protocols, in: *Proceedings of ACM SIGCOMM '90* (September 1990) pp. 201–208.

[13] C. Davis, P. Vixie, T. Goodwin and I. Dickinson, A means for expressing location information in the domain name system, RFC-1876 (January 1996).

[14] Extensible Markup language homepage, http://www.w3c.org/xml (1998).

[15] A. Fox, E. Brewer, S. Gribble and E. Amir, Adapting to network and client variability via on-demand dynamic transcoding, ASPLOS (1996).

[16] A. Fox and S.D. Gribble, Security on the move: indirect authentication using Kerberos, in: *Proceedings of the 2nd ACM International Conference on Mobile Computing and Networking* (1996) pp. 10–12.

[17] I. Goldberg, D. Wagner, R. Thomas and E. Brewer, A secure environment for untrusted Helper applications: confining the Wily Hacker, in: *Proc. of the 6th USENIX Security Symposium* (1996).

[18] J.D. Guyton and M.F. Schwartz, Experiences with a survey tool for discovering network time protocol servers, in: *Proc. of the USENIX Summer Conference* (June 1994) pp. 257–265.

[19] M. Handley, SAP: session announcement protocol, draft-ietf-mmusic-sap-00.ps, IETF (1996).

[20] M. Handley and V. Jacobson, SDP: session description protocol, draft-ietf-mmusic-sdp-03.ps, IETF (1997).

[21] A. Harter and A. Hopper, A distributed location system for the active office, IEEE Network Magazine 8(1) (January 1994).

[22] J. Ioannidis, D. Duchamp and G. Maguire, IP-based protocols for mobile internetworking, in: *ACM SIGCOMM Symposium on Communications, Architecture, and Protocols* (1991) pp. 235–245.

[23] A. Joseph, A. deLespinasse, J. Tauber, D. Gifford and M.F. Kaashoek, Rover: a toolkit for mobile information access, in: *Proceedings of the Fifteenth Symposium on Operating System Principles* (December 1995).

[24] F. Kaashoek, T. Pickney and J. Tauber, Dynamic documents, in: *Workshop on Mobile Computing Systems and Applications* (December 1994).

[25] R.H. Katz, Wireless overlay networks, in: *Proceedings 1996 SPIE Conference on Multimedia and Networking*, San Jose, CA (January 1996).

[26] R.H. Katz, E.A. Brewer, E. Amir, H. Balakrishnan, A. Fox, S. Gribble, T. Hodes, D. Jiang, G.T. Nguyen, V. Padmanabhan and M. Stemm, The Bay Area Research Wireless Access Network (BARWAN), in: *Proceedings of the Spring COMPCON Conference* (1996).

[27] D.A. Kottman, R. Wittmann and M. Posur, Delegating remote operation execution in a mobile computing environment, Mobile Networks and Applications 1(4) (January 1997) 387–398.

[28] G. Liu and G. Maguire Jr., A class of mobile prediction algorithms for wireless mobile computing and communications, Mobile Networks and Applications 1(2) (October 1996) 113–122.

[29] S. Long, R. Kooper, G. Abowd and C. Atkenson, Rapid prototyping of mobile context-aware applications: the CyberGuide case study, in: *Proc. 2nd ACM Conf. on Mobile Computing and Networking* (November 1996) pp. 97–107.

[30] B. MacIntyre and S. Feiner, Future multimedia user interfaces, Multimedia Systems Journal 4(5) (October 1996) 250–268.

[31] S. McCanne, V. Jacobson and M. Vetterli, Receiver-driven layered multicast, in: *ACM SIGCOMM '96* (August 1996) pp. 117–130.

[32] D.L. Mills, Internet time synchronization: the network time protocol, IEEE Transactions on Communications 39(10) (1992) 1482–1493.

[33] P.V. Mockapetris and K. Dunlap, Development of the domain name system, in: *ACM SIGCOMM '88* (August 1988).

[34] Object management group homepage, http://www.omg.org/ (1998).

[35] C. Perkins, IP mobility support, RFC-2002 (October 1996).

[36] J. Saltzer, D. Reed and D. Clark, End-to-end arguments in system design, ACM Transactions on Computer Systems 2(4) (1984) 195–206.

[37] B.N. Schilit, N. Adams, R. Gold, M. Tso and R. Want, The PARCTAB mobile computing system, in: *Proceedings Fourth Workshop on Workstation Operating Systems (WWOS-IV)* IEEE (October 1993) pp. 34–39.

[38] B.N. Schilit, N.I. Adams and R. Want, Context-aware computing applications, in: *Proceedings of the Workshop on Mobile Computing Systems and Applications*, IEEE Computer Society (December 1994) pp. 85–90.

[39] B.N. Schilit and M.M. Theimer, Disseminating active map information to mobile hosts, in: IEEE Network (Sept/Oct 1994) pp. 22–32.

[40] W. Schilit, System architecture for context-aware mobile computing, Ph.D. thesis, Columbia University (1995).

[41] S. Seshan, H. Balakrishnan and R.H. Katz, Handoffs in cellular wireless networks: the Daedalus implementation and experience, Kluwer Journal on Wireless Personal Communications (January 1997).

[42] B. Smith, Eolas Technologies, Inc., Computerized Processes Unlimited, Inc., and M. Roseman, DpTcl.tcl (1996). Obtained as alpha code.

[43] M. Spreitzer and M. Theimer, Providing location information in a ubiquitous computing environment, in: *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles* (December 1993).

[44] M. Stemm, P. Gauthier, D. Harada and R.H. Katz, Reducing power consumption of network interfaces in hand-held devices, in: *The Third Workshop on Mobile Multimedia Communications* (May 1996).

[45] M. Stemm and R.H. Katz, Vertical handoffs in wireless overlay networks, ACM Mobile Networking (MONET), Special Issue on Mobile Networking in the Internet (Fall 1997).

[46] P. Sukaviriya, J. Foley and T. Griffith, A second generation user interface design environment: the model and the runtime architecture, in: *Proceedings of Human Factors in Computing Systems '93* (April 1993) pp. 375–382.

[47] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall and G. Minden, A survey of active network research, IEEE Communications Magazine (January 1997) 80–86.

[48] Universal Remote Console Communication Protocol, http://trace.wisc.edu/world/urcc (1997).

[49] J. Veizades, E. Guttman, C. Perkins and S. Kaplan, Service location protocol internet draft #17, draft-ieft-svrloc-protocol-17.txt, IETF (1997).
[50] G. Voelker and B. Bershad, Mobisaic: an information system for a mobile wireless computing environment, in: *Workshop on Mobile Computing Systems and Applications* (December 1994).
[51] T. Watson, Application design for wireless computing, in: *Workshop on Mobile Computing Systems and Applications*, IEEE Computer Society (December 1994).
[52] M. Weiser, Some computer science issues in ubiquitous computing, Communication of the ACM 36(7) (July 1993).
[53] K.R. Wood, T. Richardson, F. Bennett, A. Harter and A. Hopper, Global teleporting with Java: toward ubiquitous personalized computing, IEEE Computer 30(2) (1997) 53–59.

Todd Hodes is a Ph.D. student in computer science in the Department of Electrical Engineering and Computer Sciences at UC Berkeley. He received the bachelors of science with high honors in both computer science and applied mathematics from the University of Virginia in 1994, and his masters of science in computer science from Berkeley in 1997. His current research focuses on interaction with localized services from wirelessly-connected mobile devices. This involves allowing on-the-fly discovery of interfaces through a scalable wide-area resource discovery system and development of a component object framework that integrates method invocation interface specifications and user interface information.
E-mail: hodes@cs.berkeley.edu

Randy H. Katz received his A.B. degree, with highest honors, in computer science and mathematics from Cornell University in 1976. He received the M.S. and Ph.D. degrees in computer science from the University of California, Berkeley, in 1978 and 1980, respectively. After a year in industry, he joined the Computer Sciences Department of the University of Wisconsin-Madison as an Assistant Professor. He moved to the University of California, Berkeley, in 1983, where he now holds the United Microelectronics Corporation Distinguished Professorship in Electrical Engineering and Computer Science. He also serves as the first computer scientist to chair the EECS Department. Professor Katz is a leading researcher in computer system design and implementation. His research experience has spanned numerous disciplines. He has written over 140 technical publications on CAD, database management, multiprocessor architectures, high performance storage systems, and video server architectures. He is also the author of the #1 best selling textbook on introductory hardware design, Contemporary Logic Design. Katz's recent research has focused on wireless communications, mobile computing applications, collaboration technology, and video archive systems. From January 1993 through December 1994, Katz was a program manager and deputy director of the Computing Systems Technology Office (now the Information Technology Office) of DARPA. One of his key achievements during his Washington service was to oversee the connection of the White House to the Internet, and to establish electronic mail accounts for the President and the Vice President at WhiteHouse.Gov.
E-mail: randy@cs.berkeley.edu